Java 8

Quick start

Robert Bachmann & Dominik Dorn

JSUG Meeting #63

Outline: What's new in Java 8

- Interface additions and lambda syntax (r)
- Library additions (r)
- Nashorn (d)
- Type annotations (d)
- VM changes

Interface additions

- Can add static and default (non-abstract) methods to interfaces
  - Can not provide toString, hashCode, equals
- Functional interfaces:
  - an interface with one abstract method
  - can be implemented using the new lambda syntax

Example: Default method

```
public interface Iterable<T> {
  Iterator<T> iterator();

  default void forEach(
    Consumer<? super T> action) {
        Objects.requireNonNull(action);
        for (T t : this) {
            action.accept(t);
        }
    }
}

@FunctionalInterface
public interface Consumer<T> {
  void accept(T t);
}
```

Lambda syntax examples

```
List<String> list = Arrays.asList(”a”,”b”);

list.forEach(new Consumer<String>() { // Java 7
    public void accept(String s)
    { System.out.println(s); }
});

// Java 8
list.forEach( s -> {System.out.println(s);} );

list.forEach( s -> System.out.println(s) );

list.forEach( System.out::println );
```

java.util.function

- Consumer (T $\rightarrow$ void)
- Supplier (void $\rightarrow$ T)
- Predicate (T $\rightarrow$ boolean)
- Function (T $\rightarrow$ R) and UnaryOperator (T $\rightarrow$ T)
- BiFunction, BiPredicate, BinaryOperator
- Primitive variants: IntConsumer (int $\rightarrow$ void), …

java.util.Optional<T>

- inspired by Haskell and Scala
- catch NullPointers early
- used in map() etc.
- basically a ”Box” around an object; empty or filled

java.util.Optional<T>

```
// groovy version
String version =
    computer?.getSoundcard()?.getUSB()?
.getVersion() ?: ”UNKNOWN”;
```

java.util.Optional<T>

```
public class Computer {
  private Optional<Soundcard> soundcard;
  public Optional<Soundcard> getSoundcard() { ... }
  ...
}

public class Soundcard {
  private Optional<USB> usb;
  public Optional<USB> getUSB() { ... }

}

public class USB{
  public String getVersion(){ ... }
}
```

java.util.Optional<T>

```
SoundCard soundcard = new Soundcard();

 // throws NPE if soundcard = null
Optional<Soundcard> os1
= Optional.of(soundcard);

// returns Optional.empty(); if soundcard == null
Optional<Soundcard> os2
= Optional.ofNullable(soundcard);
```

java.util.Optional<T>

```
// filtering optionals with ”ifPresent()”
maybeSoundcard.map(Soundcard::getUSB)
      .filter(usb -> ”3.0”.equals(usb.getVersion())
      .ifPresent(() -> System.out.println(”ok”));

// same as the groovy code above
String version = computer
.flatMap(Computer::getSoundcard)
        .flatMap(Soundcard::getUSB)
        .map(USB::getVersion)
        .orElse(”UNKNOWN”);
```

Library additions

- java.time: Similar to JodaTime (JEP 150)
- java.util.Arrays.parallelSort() (JEP 103)
- java.util.Base64 (JEP 135)
- Extended SecureRandom (JEP 123)
- Methods for working with unsigned integers
- JDBC 4.2 (DateTime additions) (JEP 170)
- java.util.streams

Example: Stream (filter & map)

```
personList // List<Person>
  .stream() // Stream<Person>
  .filter(p -> p.getAge() >= 18) // Stream<Person>
  .map( p -> p.getName() ) // Stream<String>
  .forEach(System.out::println) // void
;
```

Example: Stream (map & reduce)

```
personList // List<Person>
  .stream() // Stream<Person>
  .map( p -> p.getAge() ) // Stream<Integer>
  .sum() // int
;
```

Nashorn

- New JavaScript engine
- New jjs command line utility
- See previous talk by Raphael Stary

Type annotations

- Allows to use compiler plugins to add additional behaviour during compilation
- e.g. @Nullable, @NonNull, @ReadOnly
- Checker-Framework already offers quite a few handy annotations

Type annotations cont.

```
@NotNull String str1 = ...
@Email String str2 = ...
@NotNull @NotBlank String str3 = ...

Map.@NonNull Entry = ...

new @Interned MyObject()
new @NonEmpty
@Readonly List<String>(myNonEmptyStringSet)

myObject.new @Readonly NestedClass()

void monitorTemperature() throws
@Critical TemperatureException { ... }
void authenticate() throws
@Fatal @Logged AccessDeniedException { ... }
```

JEP 120: Repeating Annotations

```
@A(1)
@A(2)
@AContainer
@AContainerContainer
foo();

// equivalent to
@AContainerContainer(
@AContainer({@A(1), @A2}),
@AContainer)
foo();


Object bla = foo();
bla.getClass()
.getAnnotationsByType(
AContainerContainer.class)
```

Other reflection changes

- Access to Parameter Names at Runtime (JEP 118)
- ...

VM Changes

- PermGen was removed
- Support for AES CPU instructions

Further reading

- http://www.oracle.com/technetwork/java/javase/8-whats-new-2157071.html
- http://docs.oracle.com/javase/tutorial/
- http://www.heise.de/developer/artikel/Was-Entwickler-mit-Java-8-erwartet-1932997.html
- http://www.oracle.com/technetwork/articles/java/java8-optional-2175753.html
- http://www.mscharhag.com/2014/02/java-8-type-annotations.html
- http://openjdk.java.net/projects/type-annotations/

Questions?

Thanks

Twitter @robertbachmann

@domdorn